## 一句话概括：

顺序访问集合对象的元素，不需要知道集合对象的底层表示。

## 补充介绍：

迭代器模式（Iterator Pattern）相信大家都已经见过不少次了，当你学习Java或者其他编程语言的开始你就会接触它，因为集合就是这种设计模式：提供一种方法顺序访问聚合对象中各个元素而又无需暴露该对象的内部表示。

这种模式将元素之间的游走交给迭代器，而不是具体的聚合对象。从而达到一种低耦合目的。

关键接口有：hasNext() 判断是否还有下一个元素， next() 下一个元素对象

## 参与角色:

1）聚合类（容器类）接口

2）具体的容器类

3）Iterator接口

4）Iterator实现类

5）元素类

## 优点：

1、它支持以不同的方式遍历一个聚合对象。 2、迭代器简化了聚合类。 3、在同一个聚合上可以有多个遍历。 4、在迭代器模式中，增加新的聚合类和迭代器类都很方便，无须修改原有代码。

## 缺点：

由于迭代器模式将存储数据和遍历数据的职责分离，增加新的聚合类需要对应增加新的迭代器类，类的个数成对增加，这在一定程度上增加了系统的复杂性。

## 使用案例或场景：

使用场景：需要遍历聚合类

案例：Java集合框架：List, Set, Map 都支持迭代

## 示例程序

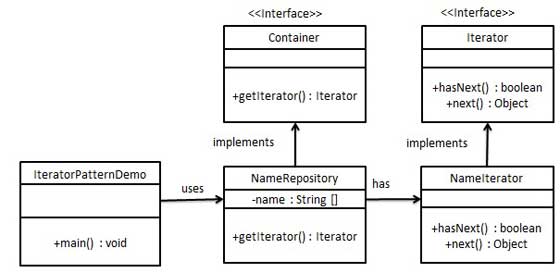
需要源码的朋友可以前往github下载：

<https://github.com/aharddreamer/chendong/tree/master/design-patterns/demo-code/design-patterns>

**程序简介**：

我们将创建一个叙述导航方法的 Iterator 接口和一个返回迭代器的 Container 接口。实现了 Container 接口的实体类将负责实现 Iterator 接口。

IteratorPatternDemo，我们的演示类使用实体类 NamesRepository 来打印 NamesRepository 中存储为集合的 Names。



**代码：**

public interface Iterator {

public boolean hasNext();

public Object next();

}

public interface Container {

public Iterator getIterator();

}

public class NameRepository implements Container {

public String names[] = {"Robert" , "John" ,"Julie" , "Lora"};

@Override

public Iterator getIterator() {

return new NameIterator();

}

private class NameIterator implements Iterator {

int index;

@Override

public boolean hasNext() {

if(index < names.length){

return true;

}

return false;

}

@Override

public Object next() {

if(this.hasNext()){

return names[index++];

}

return null;

}

}

}

public class IteratorPatternDemo {

public static void main(String[] args) {

NameRepository namesRepository = new NameRepository();

for(Iterator iter = namesRepository.getIterator(); iter.hasNext();){

String name = (String)iter.next();

System.out.println("Name : " + name);

}

}

}

**输出结果：**

Name : Robert

Name : John

Name : Julie

Name : Lora

参考：

《迭代器模式》菜鸟教程网站